Vezane liste

Strukture koje sadrže pokazivače

- Pokazivač na objekt nekog tipa smije biti član strukture.
- Dozvoljeno je da pokazivač, koji je član strukture, „pokazuje” na istu takvu strukturu, tj. da struktura sadrži pokazivač na „samu sebe”.
- Strukture koje sadrže jedan ili više članova koji su pokazivači na strukturu istog tipa (strukturu koja ih sadrži) zovu se **samoreferentne** (samoreferencirajuće, rekurzivne) strukture.

Samoreferentne strukture

Primjer:

```
struct studenti {
    char *ime;
    char *prezime;
    int ocjene[6];
    struct studenti *sljedeci;
};
```

podatkovni dio

dio za vezu

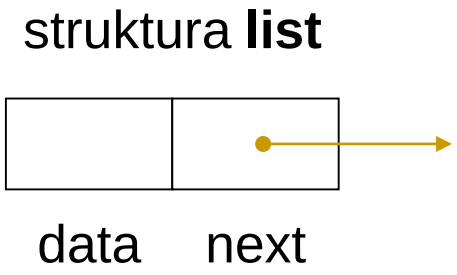
- Za članove koji su pokazivači na strukturu istog tipa obično se koriste standardna imena koja sugeriraju značenje: next, sljed, link, veza,…
- Samoreferentne strukture koristimo za implementaciju tipova podataka kao što su vezane liste i stabla.

Samoreferentne strukture (2)

Primjer:

```
struct list {
    int data;
    struct list *next
};
```

- Samoreferentne strukture uobičajeno se grafički prikazuju na sljedeći način:



Povezivanje podataka

`struct` list a, b, c;

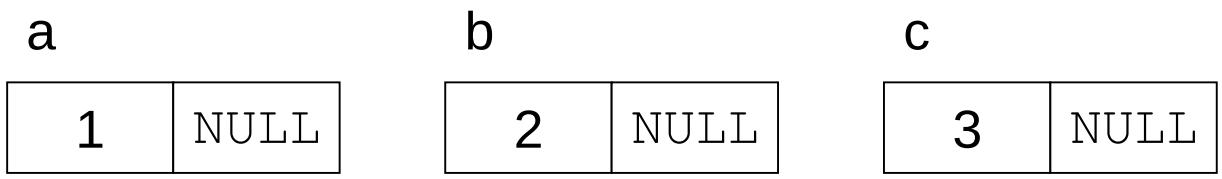
a.data = 1;

b.data = 2;

c.data = 3;

a.next = b.next = c.next = `NULL`;

- Grafički prikaz (nakon pridruživanja):

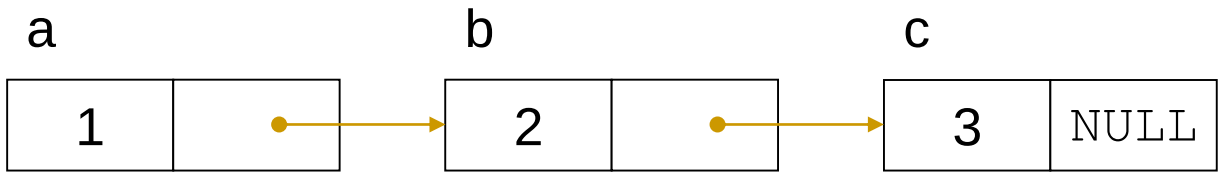


Povezivanje podataka (2)

a.next = &b;

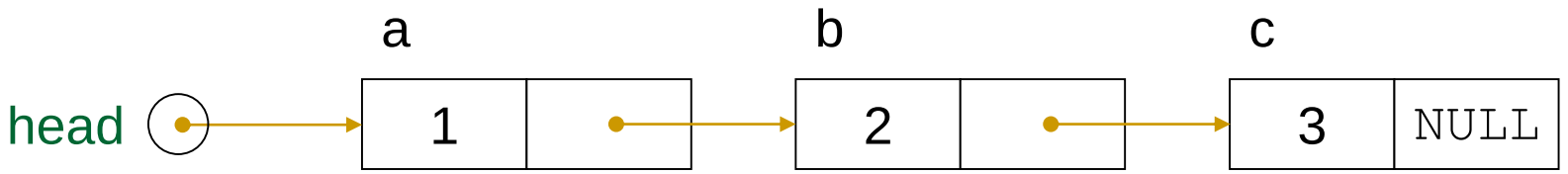
b.next = &c;

- Grafički prikaz (nakon povezivanja):



- a.next -> data ima vrijednost 2
- a.next -> next -> data ima vrijednost 3.

Vezane liste



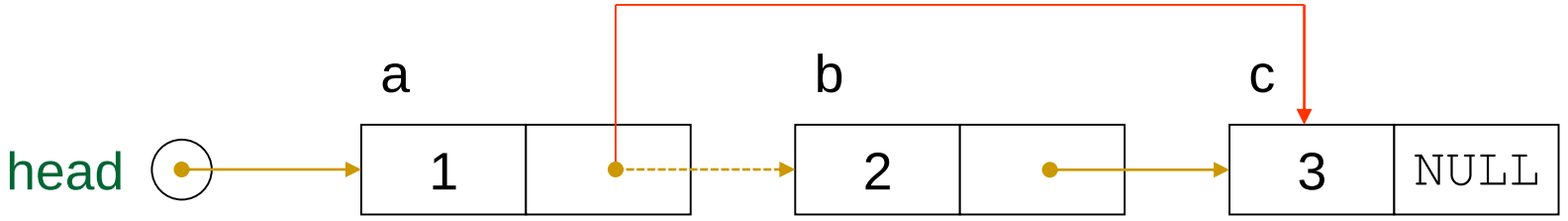
- Pokazivač head adresira prvi element liste: head = &a;

Primjer: Što će biti rezultat izvršavanja sljedećih naredbi:

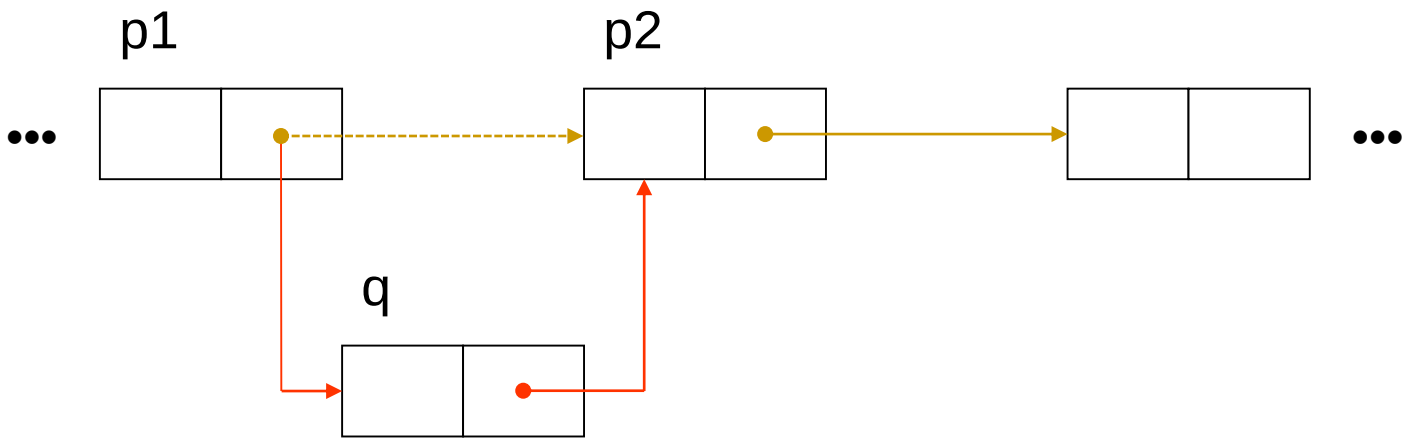
a.next = &b;

b.next = &c;

a.next = b.next;



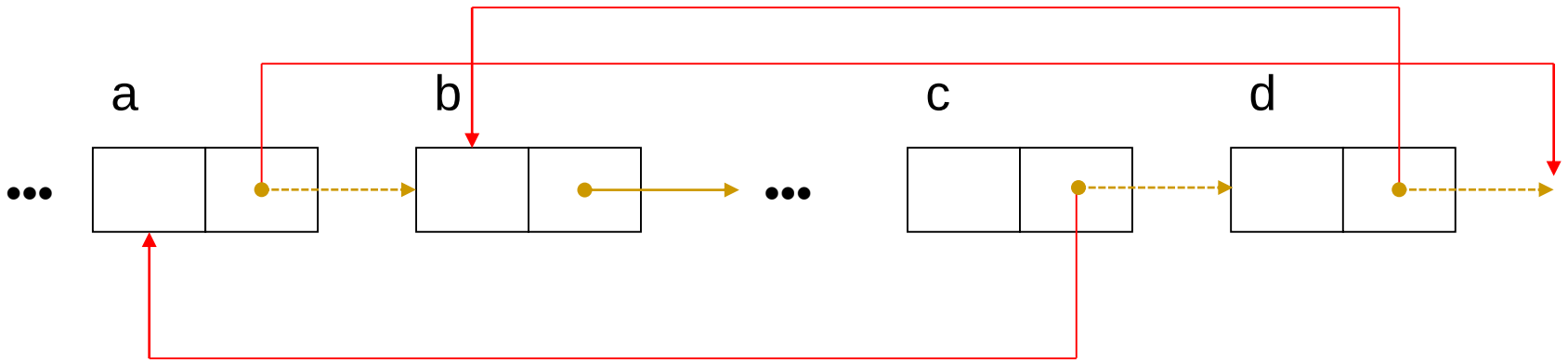
Primjer:



p1.next = &q;

q.next = &p2;

Primjer:



a.next = d.next;

d.next = &b;

c.next = &a;

Operacije nad vezanim listama

- Osnovne operacije nad listama su
  - kreiranje
  - brojanje elemenata
  - pretraživanje
  - konkatenacija (spajanje) dvaju listi
  - dodavanje novog elementa u listu
  - brisanje (izbacivanje) elemenata
  - sortiranje…
- U sljedećim primjerima implementirat ćemo neke od navedenih operacija.

Kreiranje liste

```
struct list{              /* Ovo je element liste. */
    int value;
    struct list *next;
    };

int main ()
{
   struct list *head = NULL, *list_ptr;
   int n, i;

   printf(" Učitaj broj elemenata liste:");
   scanf("%d", &n);
```

Kreiranje liste (2)

```
for (i = 0; i < n; i++){
    list_ptr = (struct list*) malloc(sizeof(struct list));
    list_ptr -> next = head;
    head = list_ptr;

    printf ("Učitaj element liste:");
    scanf ("%d", &list_ptr -> value );
}
```

Ispis liste

```
    list_ptr = head;

    while (list_ptr != NULL){
        printf ("%d\n", list_ptr -> value);
        list_ptr = list_ptr -> next;
    }
    return 0;
}
```

Primjer:

- Pretvaranje stringa u vezanu listu (string_to_list) te ispis liste (print_list) i broja elemenata liste (count) – sve navedene funkcije su rekurzivne.

```
int main ()
{
   struct list *h;

   h = string_to_list("Dobar dan");
   print_list(h);
   printf("Broj elemenata liste: %d\n", count(h));

   return 0;
}
```

Funkcija `string_to_list`

```
struct list *string_to_list(char *s)
{

    struct list *head;

    if(s[0] == '\0')
        return NULL;
    else{
        head = (struct list *) malloc(sizeof(struct list));
        head -> value = s[0];
        head -> next = string_to_list(s+1);
        return(head);
    }
}
```

Funkcija `print_list`

```
void print_list (struct list *head)
{
   if(head == NULL)
        printf("NULL\n");
   else{
        printf("%c -> ", head -> value);
        print_list(head -> next);
   }
}
```

Funkcija `count`

```
int count (struct list *head)
{
   if(head == NULL)
        return 0;
   else
        return(1 + count(head -> next));
}
```

Pretraživanje

- Napravite i iterativnu i rekurzivnu verziju funkcije

  `int` trazi (`struct` list *head, `int` *podatak*);

  koja vraća 1 ili 0 u ovisnosti o tome nalazi li se traženi *podatak* (vrijednost) u listi ili ne.

Dodavanje novog elementa

```
/* Na početak liste: */
typedef struct list* link;

link dodaj_na_pocetak (link head, link a){
    a -> next = head;
    head = a;
    return (head);
}
    ……
    a = (link)malloc(sizeof(struct list)); /* if(a == NULL) … */
    printf ("Učitaj element:");
    scanf ("%d", &a -> value );
    list_ptr = dodaj_na_pocetak (head, a);
```

Dodavanje novog elementa

```
/* Na kraj liste: */
void dodaj_na_kraj (link a){
    list_ptr = head;
    while (list_ptr -> next != NULL)
          list_ptr = list_ptr -> next;

    list_ptr -> next = a;
    a -> next = NULL;
}
    ……
    a = (link)malloc(sizeof(struct list));
    printf ("Učitaj element:");
    scanf ("%d", &a -> value );
    dodaj_na_kraj (a);
```

Spajanje lista

```
void concatenate(struct list *a, struct list *b)
{
   if(a -> next == NULL)
      a -> next = b;
   else
      concatenate(a -> next, b);
}
```

Brisanje liste

```
typedef struct list * link;

link delete(link p)
{
    link pom;
    while(p != NULL) {
        pom = p;
        p  = p -> next;
        free(pom);
    }
    return p;
}
    head = delete(head);
```